CS8592 - OBJECT ORIENTED ANALYSIS AND DESIGN

UNIT V TESTING

Object Oriented Methodologies – Software Quality Assurance – Impact of object orientation on Testing – Develop Test Cases and Test Plans.

Introduction.

Object oriented systems development is a way to develop software by building self – contained modules or objects that can be easily replaced, modified and reused. In an object–oriented environment, software is a collection of discrete objects that encapsulate their data as well as the functionality of model real–world events "objects" and emphasizes its cooperative philosophy by allocating tasks among the objects of the applications. A class is an object oriented system carefully delineates between its interface (specifications of what the class can do) and the implementation of that interface (how the class does what it does).

A method is an implementation of an object's behavior. A model is an abstract of a system constructed to understand the system prior to building or modifying it. Methodology is going to be a set of methods, models and rules for developing systems based on any set of standards. The process is defined as any operation being performed.

5.1 OBJECT ORIENTED METHODOLOGIES

Object oriented methodologies are set of methods, models, and rules for developing systems. Modeling can be done during any phase of the software life cycle .A model is a an abstraction of a phenomenon for the purpose of understanding the methodologies .Modeling provides means of communicating ideas which is easy to understand the system complexity .

Object-Oriented Methodologies are widely classified into three

1.The Rumbaugh et al. OMT (Object modeling technique)

2.The Booch methodology

3.Jacobson's methodologies

A methodology is explained as the science of methods. A method is a set of procedures in which a specific goal is approached step by step. Too min any Methodologies have been reviewed earlier stages.

- In 1986, Booch came up with the object-oriented design concept, the Booch method.
- In 1987,Sally Shlaer and Steve Mellor came up with the concept of the recursive design approach.
- In 1989, Beck and Cunningham came up with class-responsibility collaboration (CRC) cards.
- In 1990,Wirfs-Brock, Wilkerson, and Wiener came up with responsibility driven design.
- In 1991, Peter Coad and Ed Yourdon developed the Coad lightweight and prototype-oriented approach. In the same year Jim Rumbaugh led a team at the research labs of General Electric to develop the object modeling technique (OMT).
- In 1994,Ivar Jacobson introduced the concept of the use case.

These methodologies and many other forms of notational language provided system designers and architects many choices but created a much split, competitive and confusing environment. Also same basic concepts appeared in very different notations, which caused confusion among users .Hence, a new evolvement of the object oriented technologies which is called as second generation object-oriented methods.

Advantages /Charecteristics

•The Rumbaugh et al. method is well-suited for describing the object model or static structure of the system.

• The Jacobson et.al method is good for producing user-driven analysis models

• The Booch method detailed object-oriented design models

Rumbaugh et. al.'s Object Modeling Technique (OMT)

• OMT describes a method for the analysis, design, and implementation of a system using an object-oriented technique.

• Class, attributes, methods, inheritance, and association also can be expressed easily

• The dynamic behavior of objects within a system can be described using OMT Dynamic model

• Process description and consumer-producer relationships can expressed using OMT's Functional model

• OMT consists **of four phases**, which can be performed iteratively:

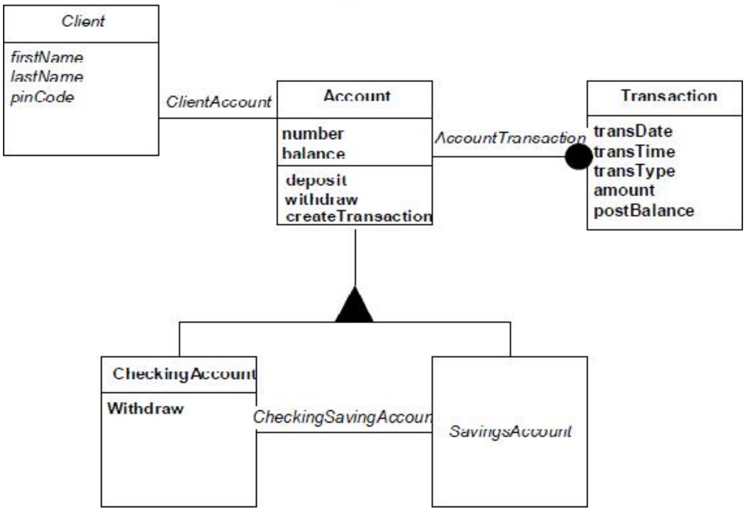
1. Analysis. The results are objects and dynamic and functional models.
2. System design. The result is a structure of the basic architecture of the system.
3. Object design. This phase produces a design document, consisting of detailed objects and dynamic and functional models.
4. Implementation. This activity produces reusable, extendible, and robust code.

• OMT separates modeling **into three different** parts:

1. An object model, presented by the object model and the data dictionary.
2. A dynamic model, presented by the state diagrams and event flow diagrams.
3. A functional model, presented by data flow and constraints.

OMT Object Model

• The object model describes the structure of objects in a system:
• Their identity , relationships to other objects, attributes, and operations
• The object model is represented graphically with an object diagram
• The object diagram contains classes interconnected by association lines



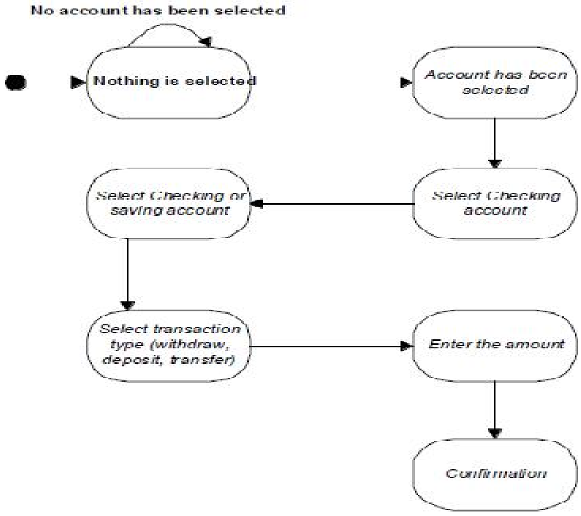
Example of an object model

- The above example provides OMT object model of a **bank system**. The boxes represent classes and the filled triangle represents specialization.
- Association between Account and Transaction is one-to-many. Since one account can have many transactions, the filled circle represents many (zero or more).

- The relationship between Client and Account classes is one-to-one. A client can have only one account and account can belong to only one person (in this model joint accounts are not considered)

OMT Dynamic Model

•OMT dynamic model depict states, transitions, events, and actions
•OMT state transition diagram is a network of states and events
• Each state receives one or more events, at which time it makes the transition to the next state.


No account has been selected
Nothing is selected
Account has been selected
Select Checking or saving account
Select Checking account
Select transaction type (withdraw, deposit, transfer)
Enter the amount
Confirmation

Example of a state transition for ATM Transaction

Here the round boxes represent states and the arrows represent transitions

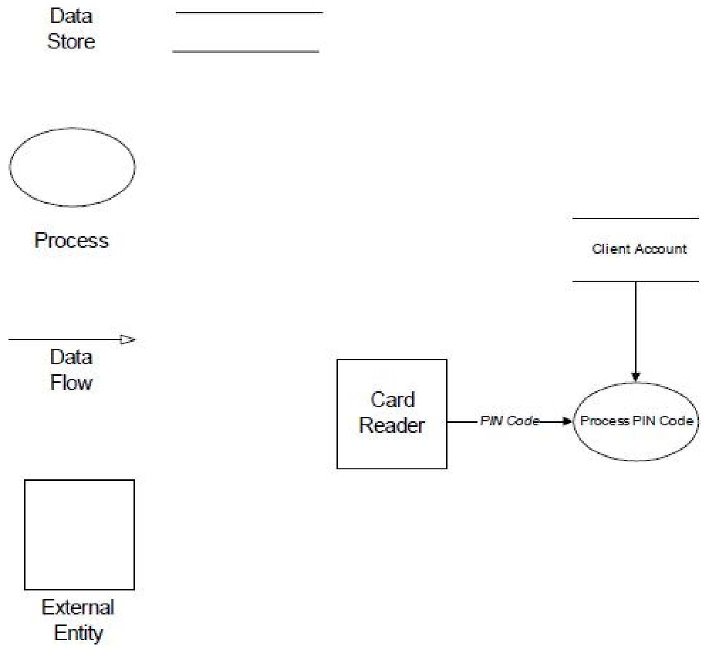
OMT Functional Model

• The OMT DFD shows the flow of data between different process in a business

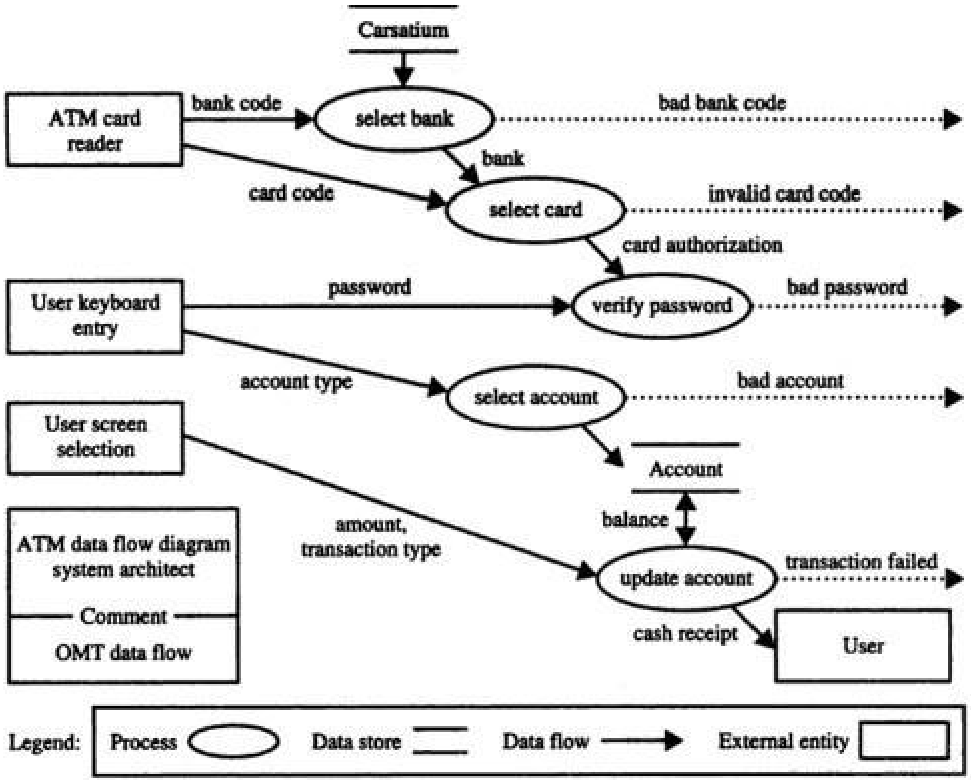
• DFD **use four primary** symbols:

• **Process** is any function being performed ; For Ex, verify password or PIN in the ATM system
• **Data flow** shows the direction of data element movement: foe Ex. PIN code
• **Data store** is a location where data are stored: for ex. Account is a data store in the ATM example
• **External entity** is a source or destination of a data element; fro ex. The ATM card Reader

On the whole , the Rumbaugh et al .OMT methodology provides one of the strongest tool sets for the analysis and design of object-oriented systems .



Example of OMT DFD of an ATM system



The above example is OMT DFD of the ATM system .The data flow lines include arrows to show the direction of data element movement .The circle represents processes. The boxes represents external entities .A data store reveals the storage of data.

The Booch Methodology

•It is a widely used object oriented method that helps us to design the system using object paradigm.

• The Booch methodology covers the analysis and design phases of systems development.

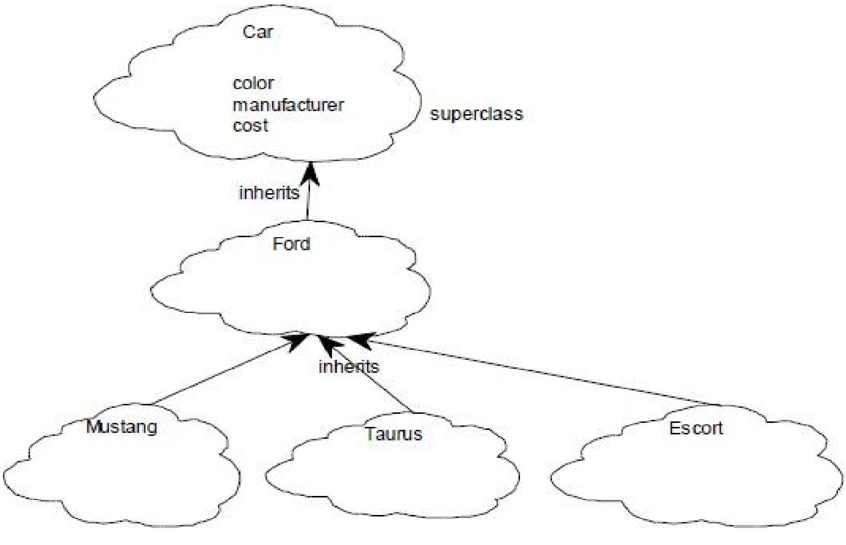
• Booch sometimes is criticized for his large set of symbols.

• You start with class and object diagram in the analysis phase and refine these diagrams in various steps.

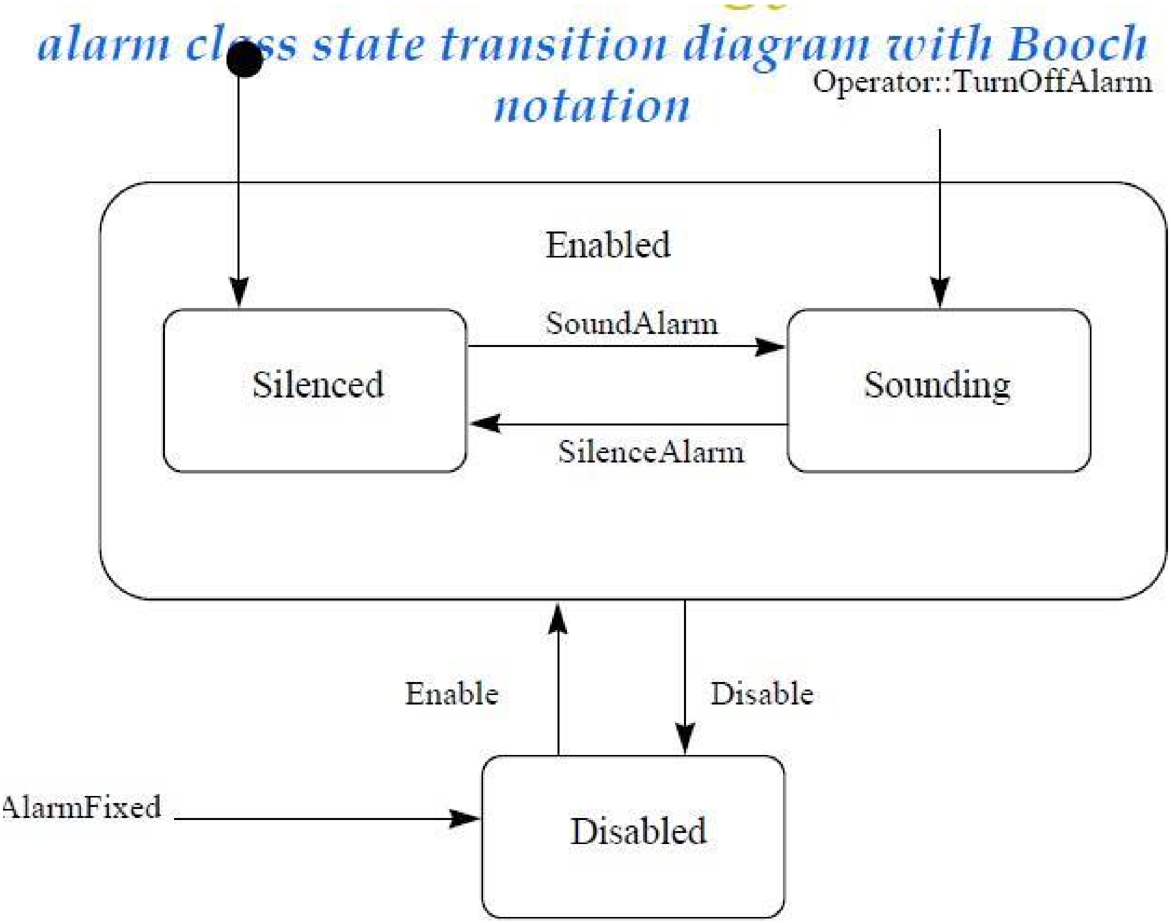
The Booch method consists of the following diagrams:

– Class diagrams
– Object diagrams
– State transition diagrams
– Module diagrams
– Process diagrams
– Interaction diagrams

Object Modeling using Booch Notation



Example : Alarm class state transition diagram with Booch notation.The arrows represents specialization



The Booch methodology prescribes

– A macro development process serve as a controlling framework for the micro process and can take weeks or even months. The primary concern of the macro process is technical management of the system
– A micro development process.

The macro development process consists of the following steps:

1. Conceptualization :

- you establish the core requirements of the system
- You establish a set of goals and develop a prototype to prove the concept

2. Analysis and development of the model.

Use the class diagram to describe the roles and responsibilities objects are to carry out in performing the desired behavior of the system .Also use the Object diagram to describe the desired behavior of the system in terms of scenarios or use the interaction diagram.

3. Design or create the system architecture.

In this phase, You use the class diagram to decide what class exist and how they relate to each other .Object diagram to used to regulate how objects collaborate. Then use module diagram to map out where each class and object should be declared. Process diagram – determine to which processor to allocate a process.

4. Evolution or implementation. – refine the system through many iterations
5. Maintenance. **-** make localized changes the the system to add new requirements and eliminate bugs.

Micro Development Process

Each macro development process has its own micro development process
• The micro process is a description of the day to- day activities by a single or small group of
s/w developers
• The micro development process consists of the following steps:
1. Identify classes and objects.
2. Identify class and object semantics.
3. Identify class and object relationships.
4. Identify class and object interfaces and implementation.

The Jacobson et al. Methodologies

• The Jacobson et al. methodologies (e.g., OOBE, OOSE, and Objectory) cover the entire life cycle and stress traceability between the different phases both forward and backward. This traceability enables reuse of analysis and design work, possibly much bigger factors in the reduction of development time than reuse of code.

Use Cases
• Use cases are scenarios for understanding system requirements.
• A use case is an interaction between users and a system.
• The use-case model captures the goal of the user and the responsibility of the system to its users.

The use case description must contain:
– How and when the use case begins and ends.

– The interaction between the use case and its actors, including when the interaction occurs and what is exchanged. How and when the use case will store data in the system.

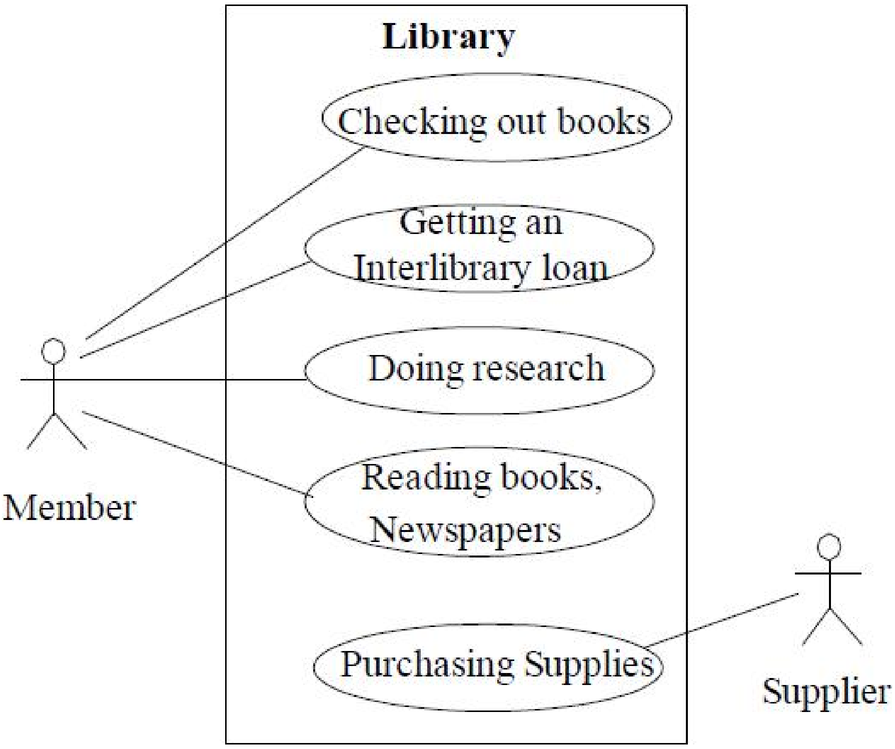
– Exceptions to the flow of events.

- Every single use case should describe one main flow events
- An exceptional or additional flow of events could be added
- The exceptional use case extends another use case to include the additional one
- The use-case model employs extends and uses relationships
- The extends relationship is used when you have one use case that is similar to another use case

The uses relationships reuse common behavior in different use cases

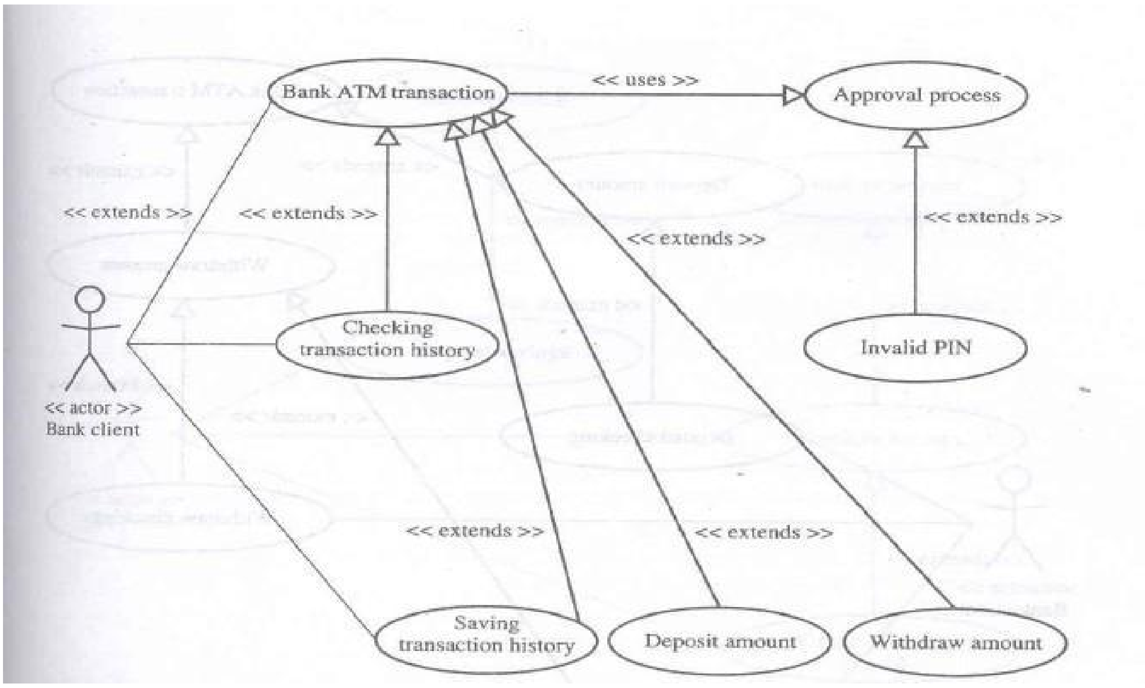
• Use cases could be viewed as a concrete or abstract

• Abstract use case is not complete and has no actors that initiate it but is used by another use case.



Abstract Usecase

ATM Transaction use cases.



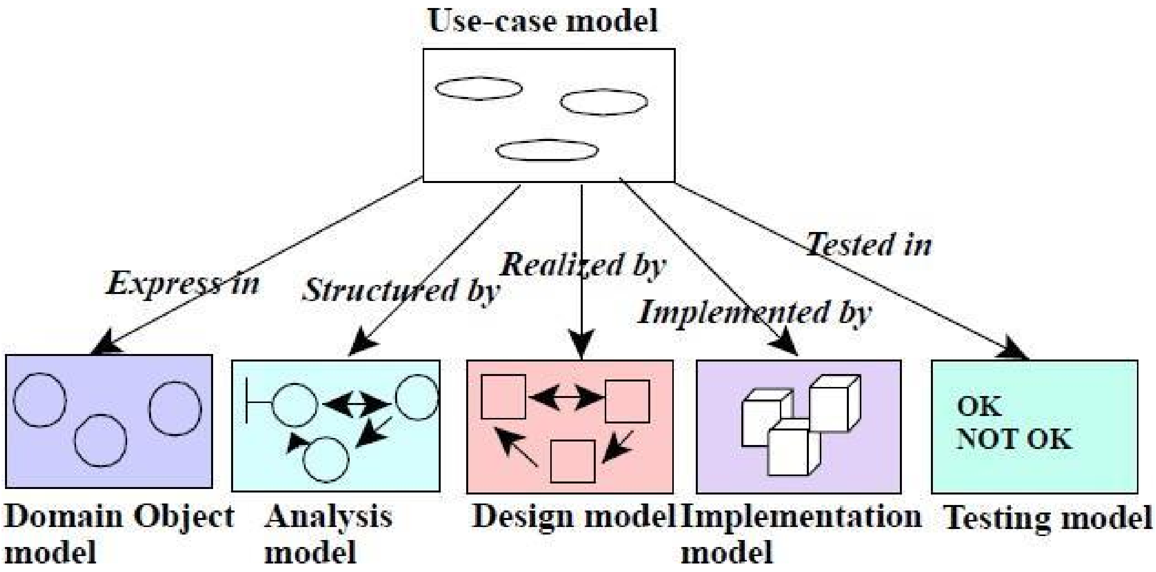
Object-Oriented Software Engineering: Objectory

- Object-oriented software engineering (OOSE), also called Objectory, is a method of object oriented development with the specific aim to fit the development of large, real-time systems. The development process, called use-case driven development, stresses that use cases are involved in several phases of the development.
- The system development method based on OOSE is a disciplined process for the industrialized development of software, based on a use-case driven design. It is an approach to object-oriented analysis and design that centers on understanding the ways in which a system actually is used.
- By organizing the analysis and design models around sequences of user interaction and actual usage scenarios, the method produces systems that are both more usable and more robust, adapting more easily to changing usage
- The maintenance of each model is specified in its associated process. A process is created when the first development project starts and is terminated when the developed system is taken out of service

Objectory is built around several different models:

- Use case model.
- defines the outside (actors) and inside(use case) of the system behavior

– Domain object model. The object of the “real” world are mapped into the domain object model
– Analysis object model.
– how the source code (implementation) should be carried out and written
– Implementation model.
– represents the implementation of the system
– Test model.
  - constitute the test plan, specifications, and reports



Object-Oriented Business Engineering (OOBE)

Object-oriented business engineering (OOBE) is object modeling at the enterprise level. Use cases again are the central vehicle for modeling, providing traceability throughout the software engineering processes.

OOBE consists of : object modeling at enterprises level
– Analysis phase

- The analysis phase defines the system to be built in terms of the problem-domain object model, the requirements model and the analysis model .This reduces complexity and promotes maintainability over the life of the system ,since the description of the system will be independent of hardware and software requirements .
- The analysis process is iterative but the requirements and the analysis models should be stable before moving on to subsequent models. Jacobson et al.

suggest that prototyping with a tool might be useful during this phase to help specify user interfaces.

– Design& Implementation phases

- The implementation environment must be identified for the design model . This include factors such as DBMS, distribution of process ,constraints due to the programming language, available component libraries and incorporation user interface tools
- It may be possible to identify implementation environment concurrently with analysis. The analysis objects that fit the current implementation environment.

– Testing phase.

Finally Jacobson describes several testing levels and techniques such as unit testing, integration testing and system testing.

Patterns

A design pattern is defined as that it identifies the key aspects of a common design sturture that make it useful for creating a reusable object-orinted design . It also identifies the participating classes and instances their roles and collaborations and the distribution of responsibilities.[Gamma,Helson,Johnson definition]

A pattern involves a general description of solution to a recurring problem bundle with various goals and constraints. But a pattern does more than just identify a solution; it also explains why the solution is needed.

• A pattern is useful information that captures the essential structure and insight of a successful family of proven solutions to a recurring problem that arises within a certain context and system of forces.

• Its help software developers resolve commonly encountered, difficult problems and a vocabulary for communicating insight and experience about these problems and their solutions.

The main idea behind using patterns is to provide documentation to help categorize and communicate about solutions to recurring problems.

• The pattern has a name to facilitate discussion and the information it represents.

A good pattern will do the following:

• It solves a problem.
Patterns capture solutions, not just abstract principles or strategies.

• It is a proven concept.
Patterns capture solutions with a track record, not theories or speculation.

• The solution is not obvious.

The best patterns generate a solution to a problem indirectly—a necessary approach for the most difficult problems of design.

• It describes a relationship.

Patterns do not just describe modules, but describe deeper system structures and mechanisms.

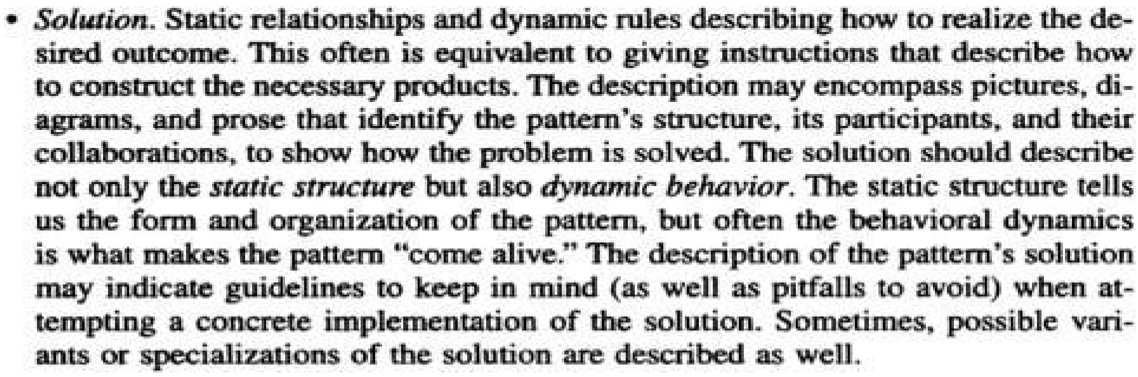
Generative and Non-Generative Patterns

- Generative patterns are the patterns that not only describe a recurring problem but also tell us how to generate something and can be observed in the resulting system architectures.
- Non-generative patterns are static and passive .They describe recurring phenomena without necessarily saying how to reproduce them.

Patterns Template

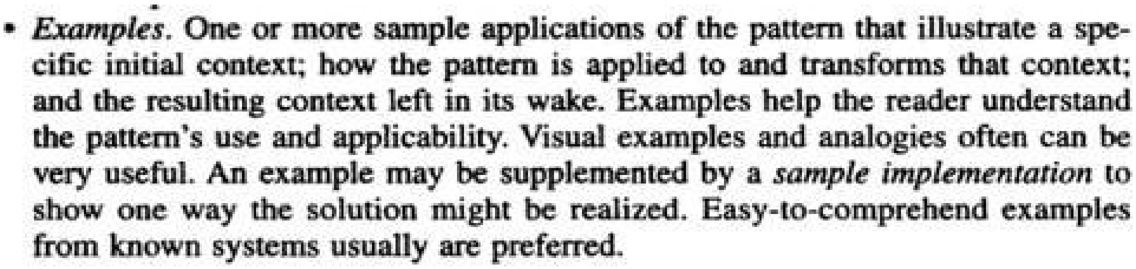
Every pattern must be expressed in form of a template which establishes a relationship between a context , a system of forces which raises in that context and a configuration which allows these forces to resolve themselves in that context. The following components should be present in a pattern template

- Name –A meaningful name .This allows us to use a singlew word or short phrase to refer a pattern and the knowledge and the structure it describes.Sometimes a pattern may have more than one commonly used or recognizable name in the literature .In this case nick names can be used .
- Problem-A statement of a problem that describes its intent: the goals and objectives it wants to reach within the given context and forces .
- Context-The preconditions under which the problem and its solution seem to recur and for which solution is desirable. This tells us about the pattern applicability.
- Forces-A description of the relevant forces and constraints and how they interact or conflict with one another and with goals to that wish to achieve. Forces reveal the intricacies of the problem and define the kinds of trade-offs that must be considered in the presences of the tension or dissonance they create. A good pattern description should fully encapsulate all the forces that have an impact on it.
- Solution



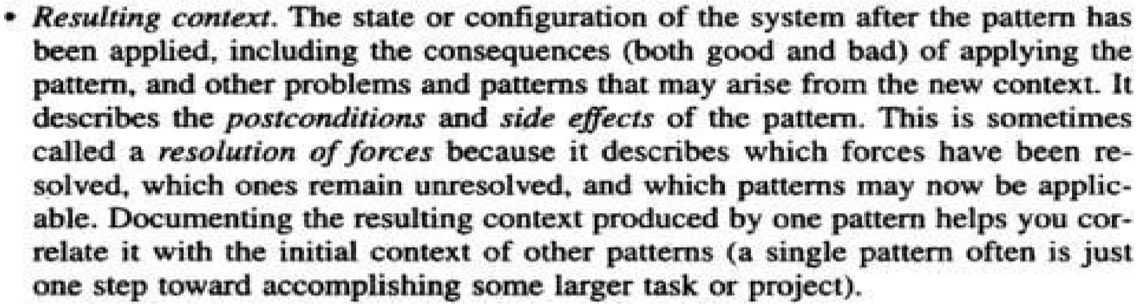
- *Solution*. Static relationships and dynamic rules describing how to realize the desired outcome. This often is equivalent to giving instructions that describe how to construct the necessary products. The description may encompass pictures, diagrams, and prose that identify the pattern's structure, its participants, and their collaborations, to show how the problem is solved. The solution should describe not only the *static structure* but also *dynamic behavior*. The static structure tells us the form and organization of the pattern, but often the behavioral dynamics is what makes the pattern "come alive." The description of the pattern's solution may indicate guidelines to keep in mind (as well as pitfalls to avoid) when attempting a concrete implementation of the solution. Sometimes, possible variants or specializations of the solution are described as well.

- Examples



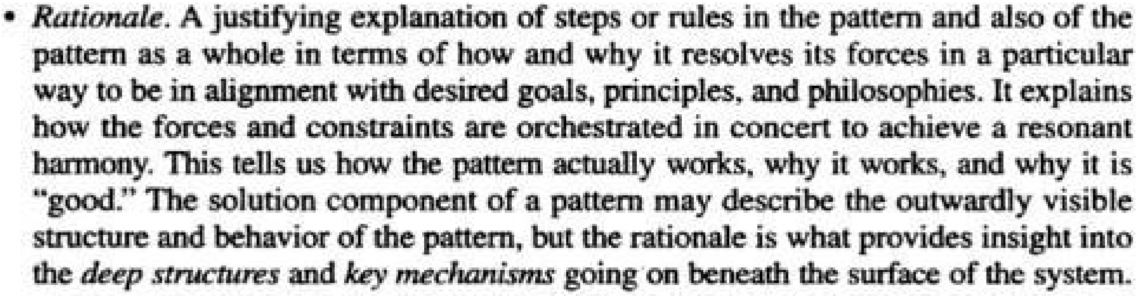
- *Examples*. One or more sample applications of the pattern that illustrate a specific initial context; how the pattern is applied to and transforms that context; and the resulting context left in its wake. Examples help the reader understand the pattern's use and applicability. Visual examples and analogies often can be very useful. An example may be supplemented by a *sample implementation* to show one way the solution might be realized. Easy-to-comprehend examples from known systems usually are preferred.

- Resulting context



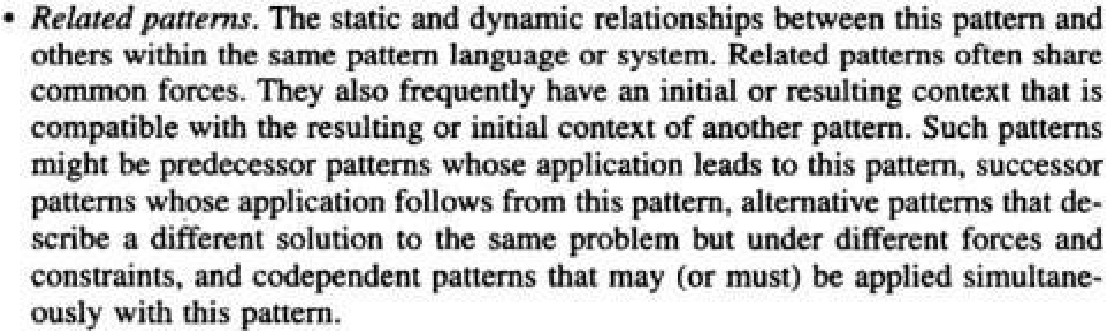
- *Resulting context*. The state or configuration of the system after the pattern has been applied, including the consequences (both good and bad) of applying the pattern, and other problems and patterns that may arise from the new context. It describes the *postconditions* and *side effects* of the pattern. This is sometimes called a *resolution of forces* because it describes which forces have been resolved, which ones remain unresolved, and which patterns may now be applicable. Documenting the resulting context produced by one pattern helps you correlate it with the initial context of other patterns (a single pattern often is just one step toward accomplishing some larger task or project).

- Rationale



- *Rationale*. A justifying explanation of steps or rules in the pattern and also of the pattern as a whole in terms of how and why it resolves its forces in a particular way to be in alignment with desired goals, principles, and philosophies. It explains how the forces and constraints are orchestrated in concert to achieve a resonant harmony. This tells us how the pattern actually works, why it works, and why it is "good." The solution component of a pattern may describe the outwardly visible structure and behavior of the pattern, but the rationale is what provides insight into the *deep structures* and *key mechanisms* going on beneath the surface of the system.

- Related Patterns



- *Related patterns*. The static and dynamic relationships between this pattern and others within the same pattern language or system. Related patterns often share common forces. They also frequently have an initial or resulting context that is compatible with the resulting or initial context of another pattern. Such patterns might be predecessor patterns whose application leads to this pattern, successor patterns whose application follows from this pattern, alternative patterns that describe a different solution to the same problem but under different forces and constraints, and codependent patterns that may (or must) be applied simultaneously with this pattern.

- Known uses-The known occurrences of the pattern and its application within existing systems .This helps validate a pattern by verifying that it indeed is a proven solution to a recurring problem .

AntiPatterns

•A pattern represents a “best practice” whereas an antipattern represents “worst practice” or a
“lesson leaned”
•Antipattern come in two verities:
• Those describe a bad solution to a problem that resulted in a bad situation
• Those describing how to get out of a bad situation and how to proceed from there to a good solution

•The pattern has a significant human component.
- All software serves human comfort or quality of life.
-The best patterns explicitly appeal to aesthetics and utility.

Capturing Patterns

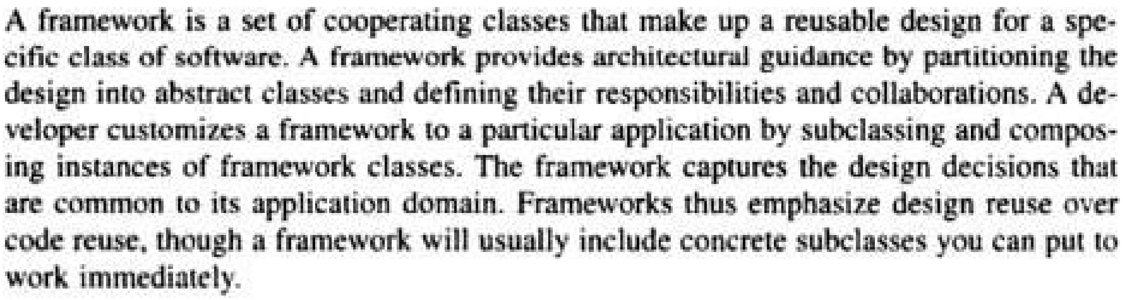
- Patterns should provide not only facts but also tell us a story that captures the experience they are trying to convey.
- A pattern should help its users comprehend existing systems, customize systems to fit user needs, and construct new systems.
- The process of looking for patterns to document is called pattern mining.

• Guidelines for capturing patterns:
– Focus on practicability.-Patterns should describe proven solutuions to recurring problems rather than the latest scientific results .
– Aggressive disregard of originality.-Pattern writers do not need to be the original inventor or discoverer of the solutions that they document.
– Non-anonymous review.-Paper submissions are shepherded rather than reviewed. It contacts the pattern authors and discusses with him or her how the patterns might be clarified or improved on
– Writers' workshops instead of presentations.-Open forums are used here to improve the patterns which are lacking
– Careful editing
.-Incorporating all the review comments and insights given by the writers workshops.

Frameworks

•A framework is a way of presenting a generic solution to a problem that can be applied to all
levels in a development.

•A single framework typically encompasses several design patterns and can be viewed as the
implementation of a system of design patterns.
A definition of object oriented software framework is given by Gamma et al.



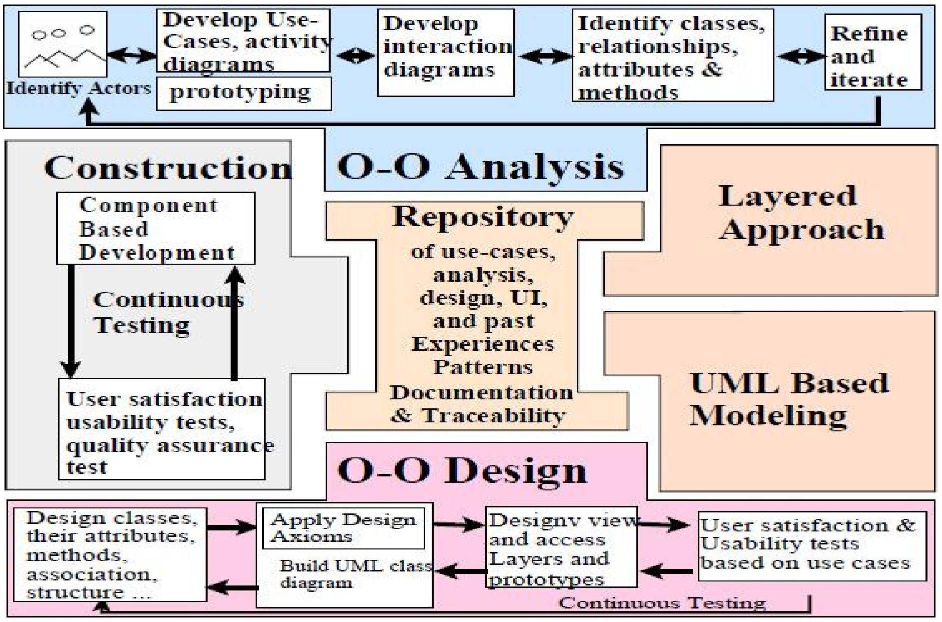
A framework is a set of cooperating classes that make up a reusable design for a specific class of software. A framework provides architectural guidance by partitioning the design into abstract classes and defining their responsibilities and collaborations. A developer customizes a framework to a particular application by subclassing and composing instances of framework classes. The framework captures the design decisions that are common to its application domain. Frameworks thus emphasize design reuse over code reuse, though a framework will usually include concrete subclasses you can put to work immediately.

Differences between Design Patterns and Frameworks

- Design patterns are more abstract than frameworks.
- Design patterns are smaller architectural elements than frameworks.
- Design patterns are less specialized than frameworks.

The Unified Approach

- The idea behind the UA is not to introduce yet another methodology.
- The main motivation here is to combine the best practices, processes, methodologies, and guidelines along with UML notations and diagrams.


Identify Actors
Develop Use-Cases, activity diagrams
prototyping
Develop interaction diagrams
Identify classes, relationships, attributes & methods
Refine and iterate
O-O Analysis
Construction
Component Based Development
Continuous Testing
User satisfaction usability tests, quality assurance test
Repository of use-cases, analysis, design, UI, and past Experiences Patterns Documentation & Traceability
Layered Approach
UML Based Modeling
O-O Design
Design classes, their attributes, methods, association, structure ...
Apply Design Axioms
Build UML class diagram
Designv view and access Layers and prototypes
User satisfaction & Usability tests based on use cases
Continuous Testing

The unified approach to software development revolves around (but is not limited to) the following processes and components.

The processes are:
– Use-case driven development.
– Object-oriented analysis.
– Object-oriented design.
– Incremental development and prototyping.
– Continuous testing.

UA Methods and Technology

• The methods and technology employed includes:
– Unified modeling language (UML) used for modeling.
– Layered approach.
– Repository for object-oriented system development patterns and frameworks.
– Promoting Component-based development.

UA Object-Oriented Analysis:

Use-Case Driven
• The use-case model captures the user requirements.
• The objects found during analysis lead us to model the classes.
• The interaction between objects provide a map for the design phase to model the relationships and designing classes.

OOA Process consists of the following steps :
1. Identify the Actors
2. Develop the simple business process model using UML activity diagram
3. Develop the Use Case
4. Develop interaction diagrams
5. Identify classes

UA Object-Oriented Design:
• Booch provides the most comprehensive object-oriented design method.
• However, Booch methods can be somewhat imposing to learn and especially tricky to figure out where to start.
• UA realizes this by combining Jacobson et al.'s analysis with Booch's design concept to create a comprehensive design process.

OOD Process consists of:

- Design classes , their attributes, methods, associations, structures and protocols, apply design axioms
- Design the Access Layer
- Design and prototype User Interface
- User satisfaction and usability Test based on the usage / Use cases

Iterative Development and Continuous Testing

- The UA encourages the integration of testing plans from day 1 of the project.
- Usage scenarios or Use Cases can become test scenarios; therefore, use cases will drive the usability testing.
- You must iterate and reiterate until, you are satisfied with the system.

Modeling Based on the Unified Modeling Language

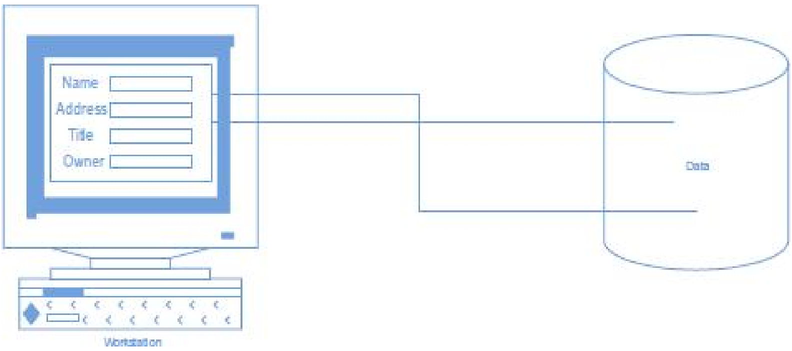
- The UA uses the unified modeling language (UML) to describe and model the analysis and design phases of system development.

The UA Proposed Repository

- The requirement, analysis, design, and implementation documents should be stored in the repository, so reports can be run on them for traceability.
- This allows us to produce designs that are traceable across requirements, analysis, design, implementation, and testing.

Two-Layer Architecture

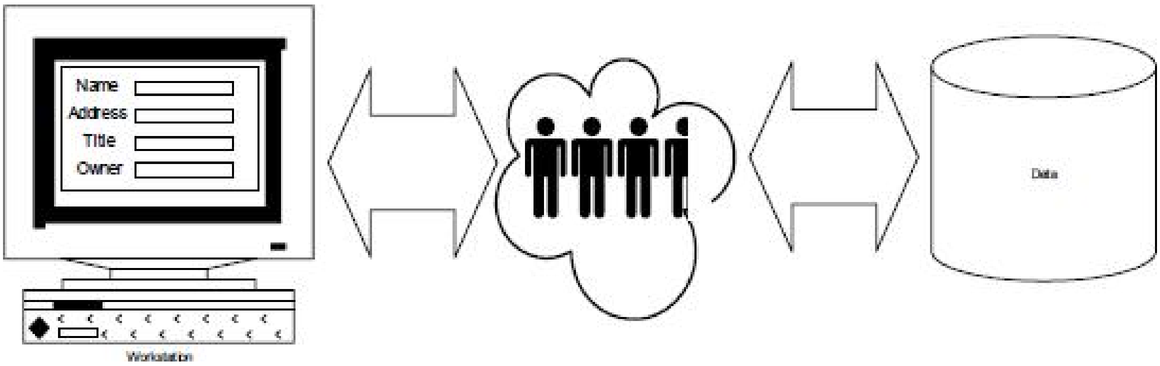
In a two-layer system, user interface screens are tied directly to the data through routines that sit directly behind the screens.



This approach results in objects that are very specialized and cannot be reused easily in other projects.

Three-Layer Architecture

• Your objects are completely independent of how:
– they are represented to the user (through an interface) or
– how they are physically stored.



User Interface layer

This layer consists of objects with which the user interacts as well as the objects needed to manage or control the interface. It is also called as a view layer. The UI interface layer objects are indentified during OOD phase .

This layer is typically responsible for two major aspects of the applications:

• Responding to user interaction-Here the user interface layer objects must be designed to translate actions by the user , such as clicking on a button or selecting from a menu ,into an appropriate response .

That response may be to open or close another interface or to send a message down into the business layer to start some business process.

• Displaying business objects.-The display of the objects is shown by using list boxes and graphs

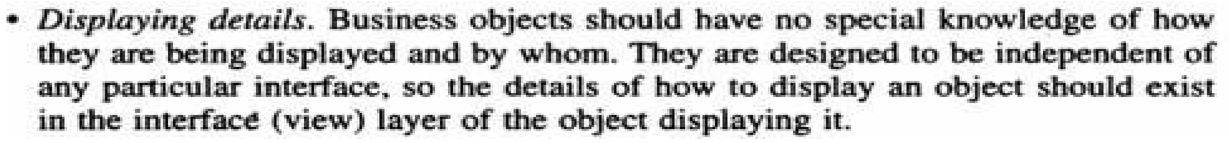
Business Layer

1.The responsibilities of the business layer are very straightforward:

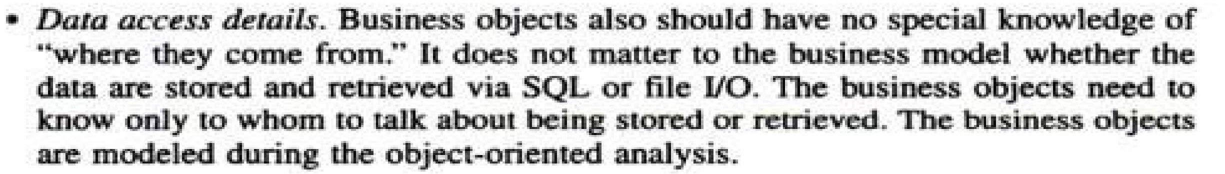
2.model the objects of the business and how they interact to accomplish the business processes.

Business Layer: Real Objects

These objects should not be responsible for:



- *Displaying details*. Business objects should have no special knowledge of how they are being displayed and by whom. They are designed to be independent of any particular interface, so the details of how to display an object should exist in the interface (view) layer of the object displaying it.

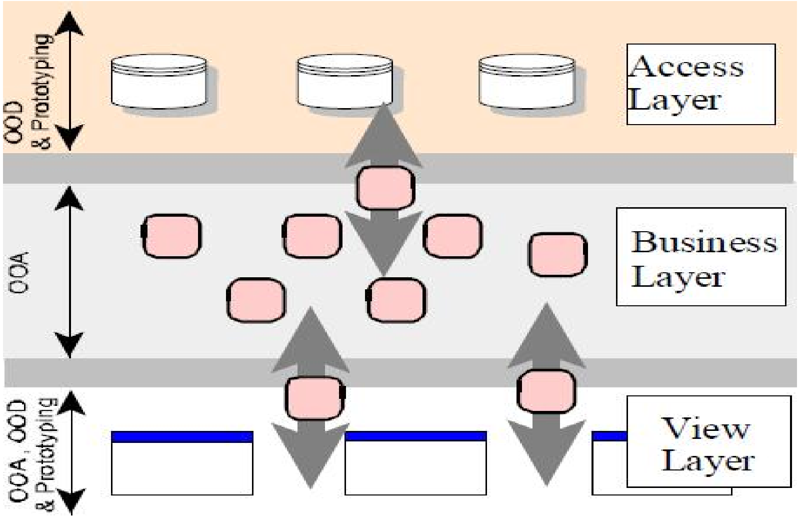


- *Data access details*. Business objects also should have no special knowledge of "where they come from." It does not matter to the business model whether the data are stored and retrieved via SQL or file I/O. The business objects need to know only to whom to talk about being stored or retrieved. The business objects are modeled during the object-oriented analysis.

Access Layer

• The access layer contains objects that know how to communicate with the place where the data actually resides,
•Whether it be a relational database, mainframe, Internet, or file.
• The access layer has two major responsibilities:
• Translate request-This layer must be able to translate any data-related requests from the business layer into the appropriate protocol for data access.(For eg . if a customer number 5333 is to be retrieved from the Database , an SQL statement is created by the access layer and execute it)
• Translate result –It translates the data retrieved back into the appropriate business objects and passes those objects back up into the business layer

Architecture for Access layer ,Business layer and view layer



5.2 SOFTWARE QUALITY ASSURANCE

The major key areas of SQA are

- Bugs and Debugging
- Testing strategies.
- The impact of an object orientation on testing.
- How to develop test cases.
- How to develop test plans.

Two issues in software quality are:

- Validation or user satisfaction
- Verification or quality assurance.

Elimination of the syntactical bug is the process of debugging. Detection and elimination of the logical bug is the process of testing.

Error Types:

•Language errors or syntax errors
•Run-time errors
•Logic errors

Identifying Bugs and Debugging

•The first step in debugging is recognizing that a bug exists.

•Sometimes it's obvious; the first time you run the application, it shows itself.

•Other bugs might not surface until a method receives a certain value, or until you take a closer look at the output

However, these steps might help:

- Selecting appropriate testing strategies
- Developing test cases and sound test plan.

Debugging Tools

•Debugging tools are a way of looking inside the program to help us determine what happens and why.

•It basically gives us a snapshot of the current state of the program.

Testing Strategies

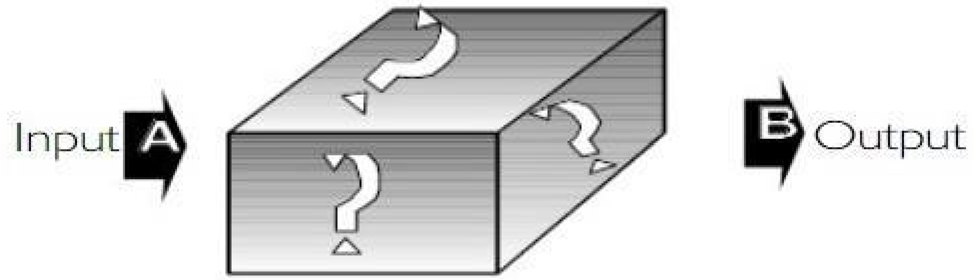
There are four types of testing strategies, These are:

Black Box Testing
White Box Testing
Top-down Testing

Bottom-up Testing

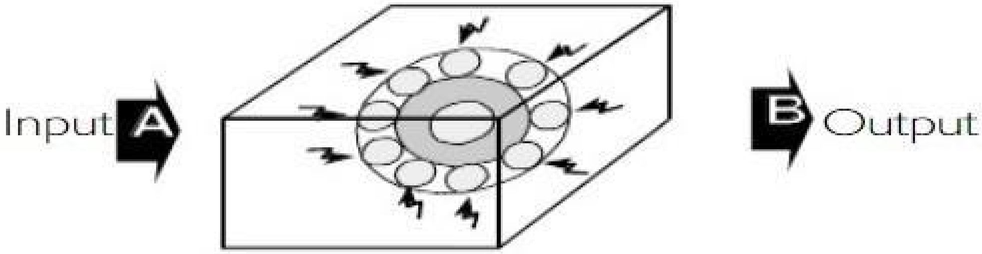
Black Box Testing

•In a black box, the test item is treated as "black" whose logic is unknown.
•All that's known is what goes in and what comes out, the input and output
•Black box test works very nicely in testing objects in an Object-Oriented environment.



White Box Testing

White box testing assumes that specific logic is important, and must be tested to guarantee system's proper functioning. This testing looks for bugs that have a low probability of execution that has been overlooked in previous investigations. The main use of this testing is error-based testing , when all level based objects are tested carefully .



One form of white box testing is called path testing
•It makes certain that each path in a program is executed at least once during testing.

Two types of path testing are:

- Statement testing coverage- The main idea of the statement testing coverage is test every statement in the objects method executing it at least once.

- Branch testing coverage –The main idea here is to perform enough tests to ensure that every branch alternative has been executed at least once under some test. It is feasible to fully test any program of considerable size.

Top-down Testing

It assumes that the main logic of the application needs more testing than supporting logic.

Bottom-up Approach

•It takes an opposite approach.

•It assumes that individual programs and modules are fully developed as standalone processes.

•These modules are tested individually, and then combined for integration testing.

System Usability & Measuring User Satisfaction

•Verification

- "Am I building the product right?"

Validation

- "Am I building the right product?"

Two main issues in software quality are
Validation or user satisfaction and
verification or quality assurance.

•The process of designing view layer classes consists of the following steps:

1. In the macro-level user interface (UI) design process, identify view layer objects.

2. In the micro-level UI, apply design rules and GUI guidelines.
3. Test usability and user satisfaction.
4. Refine and iterate the design.

Usability and User Satisfaction Testing

Two issues will be discussed:

1. Usability Testing and how to develop a plan for usability testing.
2.User Satisfaction Test and guidelines for developing a plan for user satisfaction testing.

•The International Organization for Standardization (ISO) defines usability as the effectiveness ,efficiency, and satisfaction with which a specified set of users can achieve a specified set of tasks in particular environments.

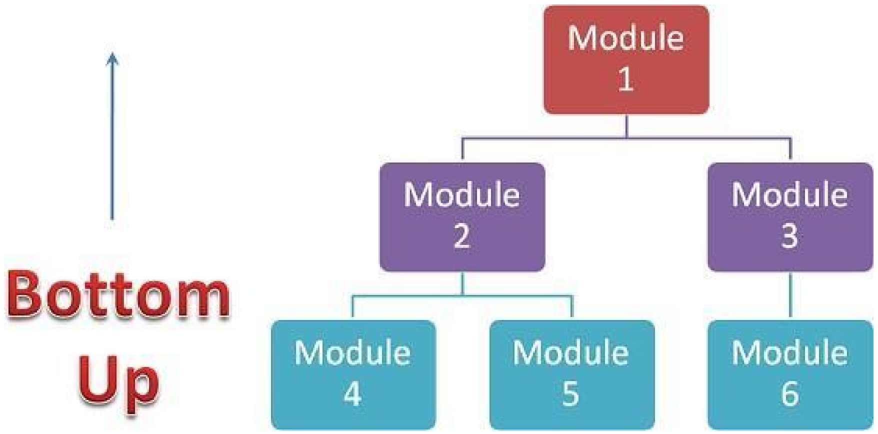
• Defining tasks. What are the tasks?

- Defining users. Who are the users?
- A means for measuring effectiveness, efficiency, and satisfaction

The phrase two sides of the same coin is helpful for describing the relationship between the
Usability and functionality of a system.

Bottom – Up Testing

It supports testing user interface and system integration. In the bottom-up strategy, each module at lower levels is tested with higher modules until all modules are tested. It takes help of Drivers for testing



Advantages:

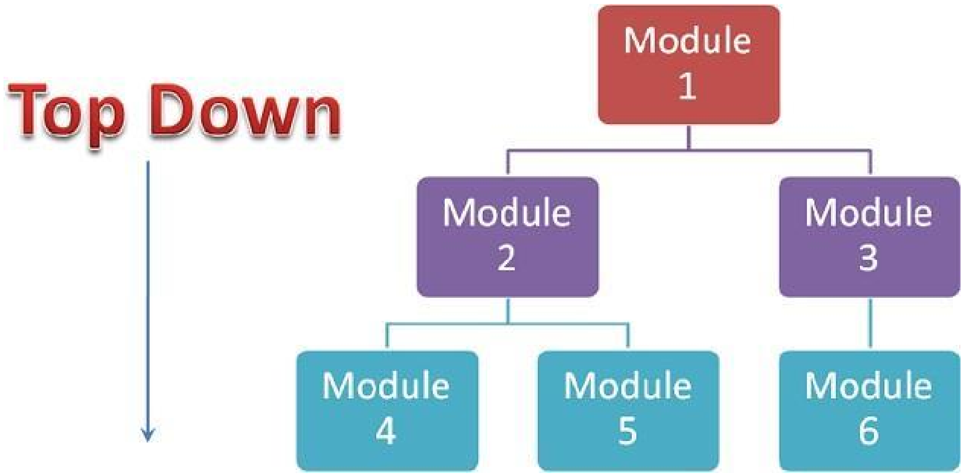
- Fault localization is easier.
- No time is wasted waiting for all modules to be developed unlike Big-bang approach

Disadvantages:

- Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.
- An early prototype is not possible

Top-down Testing:

In Top to down approach, testing takes place from top to down following the control flow of the software system. Takes help of stubs for testing. It starts with the details of the system and proceeds to higher levels by a progressive aggregation of details until they fit requirements of system.



Advantages:

- Fault Localization is easier.
- Possibility to obtain an early prototype.
- Critical Modules are tested on priority; major design flaws could be found and fixed first.

Disadvantages:

- Needs many Stubs.
- Modules at a lower level are tested inadequately**.**

5.3 IMPACT OF OBJECT ORIENTATION ON TESTING

Errors.

0 Less Plausible (not worth testing for)
0 More Plausible (worth testing for now)
0 New types of errors may appear

Impact of Inheritance on Testing.

- Does not reduce the volume of test cases
- Rather, number of interactions to be verified goes up at each level of the hierarchy
- Testing approach is essentially the same for OO oriented and Non-Object oriented environment.
- However, can reuse superclass/base class test cases
- Since OO methods are generally smaller, these are easier to test . But there are more opportunities for integration faults.

Reusability of tests.

Reusable Test Cases and Test Steps is a tool to improve re-usability and maintainability of Test Management by reducing redundancy between Test Cases in projects. Often the Test scenarios require that some Test Cases and Test Steps contain repeated or similar actions performed during a Testing cycle.

The models used for analysis and design should be used for testing at the same time. The class diagram describes relationship between objects .which is a useful information form testing .Also it shows the inheritance structure which is important information for error-based testing.

Error based testing

Error based testing techniques search a given class's method for particular clues of interests, and then describe how these clues should be tested.

Usability testing

Measures the ease of use as well as the degree of comfort and satisfaction users have with the software.

•Usability testing must begin with defining the target audience and test goals.

•Run a pilot test to work out the bugs of the tasks to be tested.

•Make certain the task scenarios, prototype, and test equipment work smoothly.

Guidelines for Developing Usability Testing

—Focus groups" are helpful for generating initial ideas or trying out new ideas.
It requires a moderator who directs the discussion about aspects of a task or design

•Apply usability testing early and often.

•Include all of software‘s components in the test.

•The testing doesn‘t need to be very expensive, a tape recorder, stopwatch, notepad and an office can produce excellent results.

•Tests need not involve many subjects.

•More typically, quick, iterative tests with a small, well-targeted sample of 6 to 10 participants can identify 80– 90 percent of most design problems.

•Focus on tasks, not features.

•Remember that your customers will use features within the context of particular tasks.

•Make participants feel comfortable by explaining the testing process.

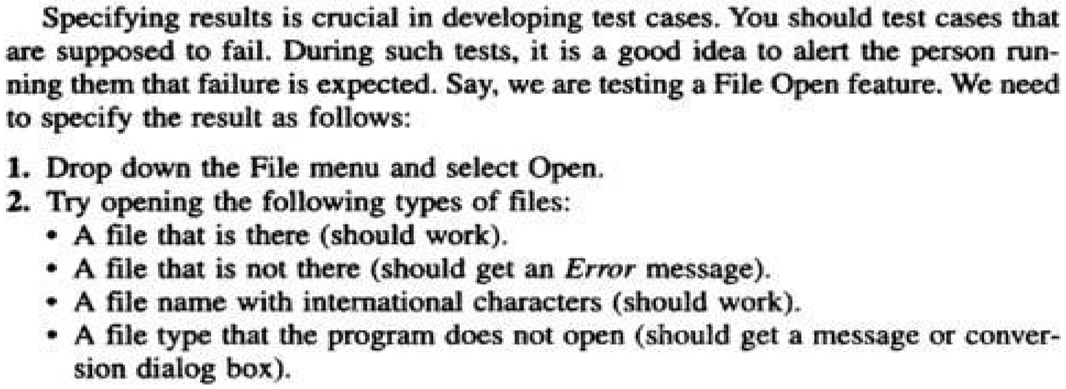
•Emphasize that you are testing the software, not the participants.

•If they become confused or frustrated, it is not a reflection on them.
•Do not interrupt participants during a test.
•If they need help, begin with general hints before moving to specific advice.
•Keep in mind that less intervention usually yields better results.
•Record the test results using a portable tape recorder, or better, a video camera.
•You may also want to follow up the session with the user satisfaction test.
•The test is inexpensive, easy to use and it is educational to those who administrate it and those who fill it out. Even if the results may never be summarized, or filled out, the process of creating the test itself will provide us with useful information.

5.4 TEST CASES

A test case is a set of What – if questions. To test a system you must construct some best input cases, that describe how the output will look. Next, perform the tests and compare the
outcome with the expected output.

Myer's (objective of testing)
Testing is a process of executing a program with the intent of finding errors.
Good test case.That has a high probability of finding an as – yet – undiscovered error.
Successful test case That detects an as – yet – undiscovered error.



Specifying results is crucial in developing test cases. You should test cases that are supposed to fail. During such tests, it is a good idea to alert the person running them that failure is expected. Say, we are testing a File Open feature. We need to specify the result as follows:

1. Drop down the File menu and select Open.
2. Try opening the following types of files:
   - A file that is there (should work).
   - A file that is not there (should get an *Error* message).
   - A file name with international characters (should work).
   - A file type that the program does not open (should get a message or conversion dialog box).

Guidelines for Developing quality assurance test cases.

Freedman and Thomas have developed guidelines that have been adopted for the UA:

- Describe which feature or service your test attempts to cover.
- If the test case is based on a use case, it is good idea to refer to the use-case name.
- Specify what you are testing and which particular feature.
- test the normal use of the object methods.

- test the abnormal but reasonable use of the objects methods.
- test the boundary conditions.
- Test objects interactions and the messages sent among them.
- Attempting to reach agreement on answers generally will raise other what-if questions.
- The internal quality of the software, such as its reusability and extensibility, should be assessed as well.

5.5 TEST PLAN

⋆ A Test plan is developed to detect and identify potential problems before delivering the
software to its users.
⋆ A test plan offers a road map.
⋆ A dreaded and frequently overlooked activity in software development.

Steps

- Objectives of the test.- create the objectives and describes how to achieve them
- Development of a test case- develop test case, both input and expected output.
- Test analysis.- This step involves the examination of the test output and the documentations of the test results

Regression Testing.- All passed tests should be repeated with the revised program, called "Regression". This can discover errors introduced during the debugging process. When sufficient testing is believed to have been conducted, this fact should be reported, and testing to this specific product is complete

Beta Testing.

Beta Testing can be defined as the second stage of testing any product before release where a sample of the released product with minimum features and characteristics is being given to the intended audience for trying out or temporarily using the product.

Unlike an alpha test, the beta test is being carried out by real users in the real environment. This allows the targeted customers to dive into the product's design, working, interface, functionality, etc.

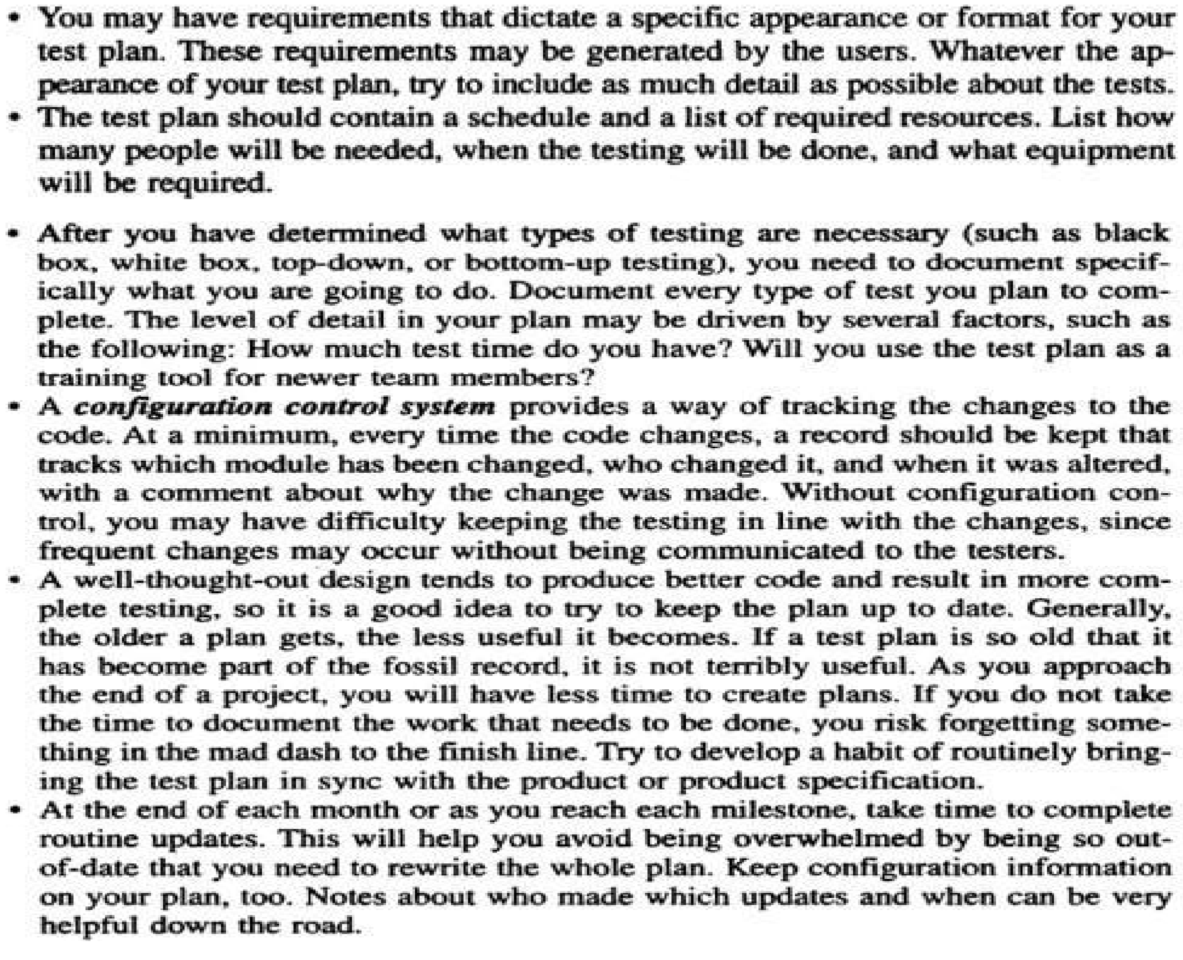
Alpha Testing.

Alpha Testing can be defined as a form of acceptance testing which is carried out for identifying various types of issues or bugs before publishing the build or executable of software public or market. This test type focuses on the real users through black box and white box testing techniques. The focus remains on the task which a general user might want or experience.

Alpha testing any product is done when product development is on the verge of completion. Slight changes in design can be made after conducting the alpha test. This testing methodology is performed in lab surroundings by the developers.
Here developers see things in the software from users point and try to detect the problems. These testers are internal company or organization's employees or may be a part of the testing team. Alpha testing is done early at the end of software development before beta testing.

Guidelines (for preparing test plan)

- Specify Requirements generated by user.
- Specify Schedule and resources.
- Determine the testing strategy.
- Configuration Control System.
- Keep the plan up to date.
- At the end of each milestone, fill routine updates**.**



- **You may have requirements that dictate a specific appearance or format for your test plan. These requirements may be generated by the users. Whatever the appearance of your test plan, try to include as much detail as possible about the tests.**
- **The test plan should contain a schedule and a list of required resources. List how many people will be needed, when the testing will be done, and what equipment will be required.**
- After you have determined what types of testing are necessary (such as black box, white box, top-down, or bottom-up testing), you need to document specifically what you are going to do. Document every type of test you plan to complete. The level of detail in your plan may be driven by several factors, such as the following: How much test time do you have? Will you use the test plan as a training tool for newer team members?
- A ***configuration control system*** provides a way of tracking the changes to the code. At a minimum, every time the code changes, a record should be kept that tracks which module has been changed, who changed it, and when it was altered, with a comment about why the change was made. Without configuration control, you may have difficulty keeping the testing in line with the changes, since frequent changes may occur without being communicated to the testers.
- A well-thought-out design tends to produce better code and result in more complete testing, so it is a good idea to try to keep the plan up to date. Generally, the older a plan gets, the less useful it becomes. If a test plan is so old that it has become part of the fossil record, it is not terribly useful. As you approach the end of a project, you will have less time to create plans. If you do not take the time to document the work that needs to be done, you risk forgetting something in the mad dash to the finish line. Try to develop a habit of routinely bringing the test plan in sync with the product or product specification.
- At the end of each month or as you reach each milestone, take time to complete routine updates. This will help you avoid being overwhelmed by being so out-of-date that you need to rewrite the whole plan. Keep configuration information on your plan, too. Notes about who made which updates and when can be very helpful down the road.

MYER’S DEBUGGING PRINCIPLES

***Bug locating principles*.**

Think

If you reach an impasse, sleep on it.

If the impasse remains, describe the problem to someone else. Use debugging tools.

Experimentation should be done as a last resort.

***Debugging principles*.**

Where there is one bug , there is likely to be another.

Fix the error, not just the symptom.

The probability of solution being correct drops down as the size increases.

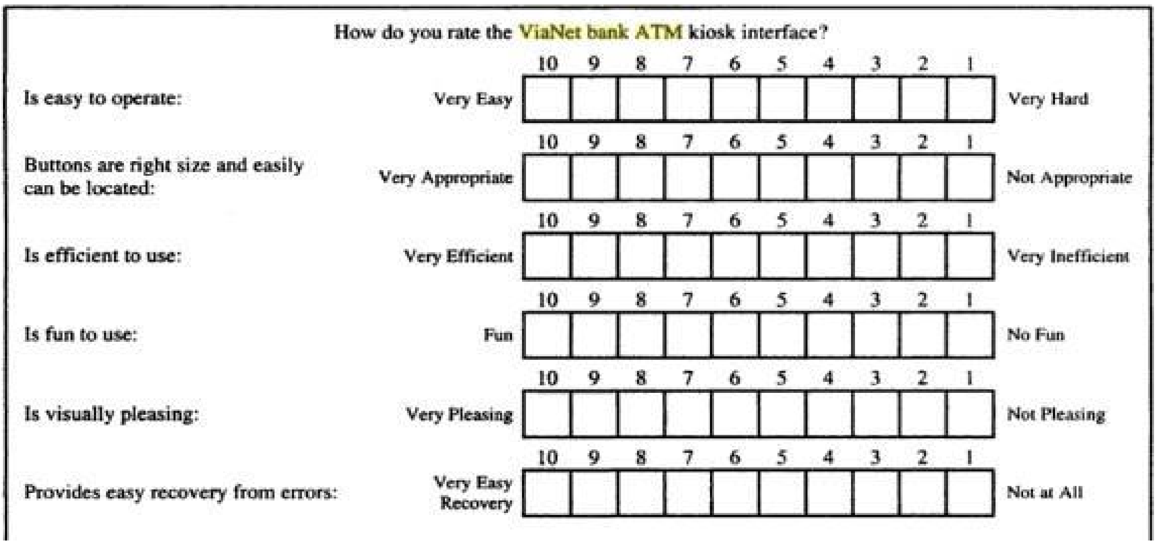
Beware of error correction, it may create new errors

Case study :

Develping Test cases for vianet ATM bank system

Test cases are derived from the following use case scenarios

1. Bank Transaction
2. Checking transaction history
3. Savings/current account
4. Deposit/Withdarw
5. valid/invalid PIN



How do you rate the ViaNet bank ATM kiosk interface?

		10	9	8	7	6	5	4	3	2	1	
Is easy to operate:	Very Easy	□	□	□	□	□	□	□	□	□	□	Very Hard
Buttons are right size and easily can be located:	Very Appropriate	□	□	□	□	□	□	□	□	□	□	Not Appropriate
Is efficient to use:	Very Efficient	□	□	□	□	□	□	□	□	□	□	Very Inefficient
Is fun to use:	Fun	□	□	□	□	□	□	□	□	□	□	No Fun
Is visually pleasing:	Very Pleasing	□	□	□	□	□	□	□	□	□	□	Not Pleasing
Provides easy recovery from errors:	Very Easy Recovery	□	□	□	□	□	□	□	□	□	□	Not at All

Example of vianet ATM system for user satisfaction test